# Approximate Query Evaluation *

Mutsumi Komuro
Hitachi Software Engineering Co.,Ltd
komuro@ori.hitachi-sk.co.jp

## Abstract

*Approximate Query Evaluation is a pattern language to extract crucial information from a given database query expression. This information will be used as an important clue to optimize the database query. Approximate Query Evaluation provides an example of an application of formal method development, where the abstract interpretation is used for the mathematical modeling. In particular a correctness proof of the optimization is given, which ensures that we never miss a record satisfying the query expression.*

## Query Estimation

... assume that you have decided to develop a database wrapper that translates a query language into the other, e.g., SQL into a query language for network database.

**How can you efficiently and correctly retrieve the requested records?**

Since the exact translation between different query languages is generally impossible, we need to use a kind of approximation or *Query Estimation*.

*Query Estimation* extracts the most important information that is necessary to retrieve the requested records. For example a network database system provides a special mechanism called 'control key' that enables the direct access to the database. Hence *Query Estimation* extracts the information about the control key. More specifically *Query Estimation* estimates the range of the control key. Since *Query Estimation* evaluates the query without dealing with detailed information, it can be fairly efficient.

*Query Estimation* should be correct in the sense that it never misses a necessary record as a result of the estimation. For example in the case of network database *Query Estimation* gives a range $R$ that includes the actual range $R_c$ of the control key of the requested records. That is, we should have the relation $R \supseteq R_c$ but not necessarily $R = R_c$.

It is extremely difficult to give a correctness proof for a real world system, even if the number of properties to be proved is very limited. We use the following strategy to overcome this difficulty. First begin with *The Initial Model* and then perform *Reification* and finally optimize the result by *Sequencing*. Since *Initial Model* has a simple mathematical structure, it is fairly easy to give a correctness proof for this model. Then we prove that each step of the above refinement preserves the correctness.

**Therefore:**

Evaluate the query by *Query Estimation* and access the database according to the result of the evaluation. Re-evaluate the query condition with each of the retrieved record and discard it if it does not satisfy the query condition.

## Initial Model

... now *Query Estimation* has defined a framework for database access through database wrapper. You need to prove the correctness of the system resulting from this framework.

### How can the correctness be proved?

*Query Estimation* is supposed to extract the information we are interested in from the query expressions. Other pieces of information are unimportant or undecidable without accessing to the database. We can model *Query Estimation* as a map $\mathcal{E} : Q \rightarrow D$ from the set of query expressions $Q$ to an evaluation domain $D$.

It is worthwhile noting that at this stage we only think about what *Query Estimation* should be but do not worry about how it can be implemented. In other words we give the specification of *Query Estimation*. Furthermore we avoid considering real world constraints about the system in order to make the model simpler.

In order to formulate the statement of correctness proof, we need to make it clear what we mean by $\mathcal{E} : Q \rightarrow D$. That is, we need to do the following.

1. Determine the range of expressions $Q$ to evaluate.

   The first thing we should do is to make it clear what kind of expressions we should evaluate. This depends on the query language and the application. Typically the range is as follows. Let $x$ and $y$ be variables, $c$ be constant, op be a binary operator of comparison i.e., one of $=, <, >, \leq$, or $\geq$, the expression is either (i) a simple expression of the form $xopy$, $xopc$, or (ii) a compound expression combining several simple expressions with a logical operator $'and'$, $'or'$, or $'not'$.

2. Decide the evaluation domain $D$.

   Create a domain to represent the information we are interested in. Make a special value called $undefined$ in the domain and map the unimportant or undecidable pieces of the information into this value. In the case of network database we can take the set of all the subsets of non-negative integers $N_0 = \{n : integer | n \geq 0\}$ as the domain, because we are interested in the possible range of control key.

3. Define fundamental operations.

   Define the fundamental operations on $D$ so that we can represent *Query Estimation*. In particular we need to define the fundamental operations on the $undefined$ value. Operations on $Q$ are mapped to these fundamental operations on $D$.

### Therefore:

Give a correctness proof of the initial model of *Query Estimate* thus defined. The correctness can be formulated as in the following Proposition. Let $r(x)$ be a record in the network database with the control key $x$. Let $Satisfy(r, P)$ be a logical expression to detect whether $P$ satisfies $r$.

**Proposition 1** $\mathcal{E}[P] \supseteq \{x \in [0, \infty) | Satisfy(r(x), P) = true\}$

Once you have proved the correctness of the *Initial Model*, it can be refined into design level by *Reification* and then into implementation level by *Sequencing* with the correctness preserved at each level.

## Example of Initial Model

In the case of network database we choose the fundamental operations to be the following.

1. Inclusion '$\subseteq$',
2. Union '$\bigcup$',
3. Intersection '$\bigcap$',
4. Set difference '$\setminus$'.

The last three operations will be used to represent logical operations $'or', 'and', 'not'$ respectively. These operations are defined to act on the undefined value $U$ as follows: Let $A$ be any subset of $[0,\infty)$. We define $U \bigcap A = A \bigcap U = A$, $U \bigcup A = A \bigcup U = U$, $[0,\infty) \setminus U = U$. Similarly for the inclusion relation $\subseteq$, we define $A \subseteq U$. In particular $[0,\infty) \subseteq U$ holds.

Using these operations, *Query Estimation* $\mathcal{E}$ is defined by the structural induction as follows: Let $P$ be a logical expression in a database query and $C$ be the control key.

1. When $P$ does not involve a logical operator,
   (a) If $P$ contains a variable other than $C$, then we define $\mathcal{E}[P] = U$.
   (b) If $P$ does not contain a variable other than $C$, we consider $P$ as a function of $C$ and denote it as $P(C)$. We define $\mathcal{E}[P(C)] = \{x \in [0,\infty) | P(x) holds.\}$.
   (c) If $P$ is of the form $not(P')$ where $P'$ does not contain any logical operator, then we define $\mathcal{E}[P] = [0,\infty) \setminus \mathcal{E}[P(C)]$.
2. If $P$ contains a logical operator but not of the form as in (2), then we define
   - $\mathcal{E}[P1 and P2] = \mathcal{E}[P1] \bigcap \mathcal{E}[P2]$,
   - $\mathcal{E}[P1 or P2] = \mathcal{E}[P1] \bigcup \mathcal{E}[P2]$,
   - $\mathcal{E}[not(P1 and P2)] = \mathcal{E}[not(P1)] \bigcup \mathcal{E}[not(P2)]$,
   - $\mathcal{E}[not(P1 or P2)] = \mathcal{E}[not(P1)] \bigcap \mathcal{E}[not(P2)]$.

Proposition 1 is easily proved by the structural induction (see [Komuro] for detail). Note that the range of $\mathcal{E}$ is $[0,\infty) \bigcup \{U\}$. Let $p$ be the projection $p : [0,\infty) \bigcup \{U\} \rightarrow [0,\infty)$ and $\mathcal{E}'$ be the composite map $p \circ \mathcal{E}$. Since $U$ ncludes any subset of $[0,\infty)$ by the definition of the inclusion relation, the above proposition holds if we replace $\mathcal{E}$ to $\mathcal{E}'$.

## Reification

... now you have got the *Initial Model*. You need to design the architecture to realize this model.

**How can you give a design preserving the correctness?**

What we have done so far is giving the specification of *Query Estimation* through abstract interpretation [Hughes]. You need to choose data type and algorithm, or in other words class architecture and method, to realize the specification. There are several issues you need to consider.

1. Separation of concern.

   The set of query expressions $Q$ is stable, whereas the evaluation domain $D$ is modified during the *Reification*. Hence you should separate the class structure of $D$ from that of $Q$.

2. Class structure and method for $Q$.

   Since $Q$ is represented as AST (abstract syntax trees) of the query expressions, its class structure can be given by Composite Pattern [GOF] so that each node of AST has an 'interpret' method as in Interpreter Pattern [GOF].

3. Class structure and methods for $D$.

   Next we will design the evaluation domain $D$. In general the evaluation domain is a collection of elements Since there are several different kinds of elements, it is natural to use Composite Pattern [GOF]. What is peculiar here is that we should distinguish covariant and contravariant subclasses, because it affects the way the fundamental operations are induced. If it is covariant, we can simply take the same fundamental operation. If it is contravariant, we need to take the dual operation. That is, the inclusion relation takes the reversed order at the contravariant classes.

   In the case of network database we can regard the control key as a variable which takes non-negative integers. It follows that the range of the control key can be modeled as a finite set of intervals in non-negative integers. An interval is a pair of left and right endpoints. We have a partial order in the set of intervals defined by the inclusion relation. On the other hand endpoints have a (partial) order by comparing as numbers. With respect to these partial orders the right endpoints are covariant, whereas the left endpoints are contravariant. The right end point is covariant in that increasing the right end point increases the range of keys while the left end point is contravariant in that increasing the left end point reduces the range of keys.

4. Introducing real world constraints.

   In *Initial Model* we did not consider the real world constraints. For example an infinite set such as non-negative integers $N_0$ might be used for modeling. This is not acceptable as a design level consideration and the maximum value might be introduced.

**Therefore:**

Refine the data and operations of *Initial Model* so that the relations and constraints given in the model are still satisfied after the *Reification*.

After you have refined *Initial Model* by *Reification*, you can realize an efficient implementation by *Sequencing*.

## Example of Reification

In the case of network database the evaluation domain $D$ consists of a finite number of intervals. And each interval is represented by a pair of endpoints, i.e., left and right endpoints. The former forms a contravariant subclass LeftEndPoint and the latter does a covariant subclass RightEndPoint.

Note that an interval also has some topological properties, such as it is open or closed. Since we use only integers for modeling the database access, it is possible to normalize the representation of the intervals so that we have only to deal with, say, closed intervals. For example, we can rewrite an open interval $(a, b)$ into a closed interval $[a + 1, b - 1]$. But a control key could have an extremely long byte length in the specification of our particular network database, which forced us to represent it as string. It follows that numerical operations such as "add 1" cannot be efficiently implemented in our representation, although comparison operations can be. Therefore we employ a modeling that distinguishes open and closed intervals, without normalizing their representations. This is one example of 'real world constraints.'

Now that we distinguish open and closed, there are four kinds of intervals depending on the topological properties around the endpoints. This may complicate the implementation of set operations. A natural solution would be to consider these topological properties are attributed to endpoints, not to the interval they consist. If an endpoint belongs to the interval, we say it is open. If not, we say it is closed.

We can define operations on endpoint subclasses so that the fundamental operations of $D$ are rephrased in terms of these operations. For example, we define a comparison operation $\leq_{left}$ for two left endpoints $lp_1$ and $lp_2$ as follows. The relation $lp1 \leq_{left} lpt2$ holds if and only if $lpt1.value < lpt2.value$ *or* $lpt1.value = lpt2.value$ *and* $(lpt1.topology = lpt2.topology$ *or* $lpt1.topology = closed$ *and* $lpt2.topology = open)$, where value and topology are the attributes of left endpoint class representing the numerical value or open/closed attribute respectively. Let $MAX$ be the maximum value introduced by another 'real world constraint' and denote the right endpoint of value $MAX$ as the same symbol for the sake of simplicity. Then we have the following proposition.

**Proposition 2** *The interval defined by the pair of end points* $(lpt_1, MAX)$ *is included in another interval* $(lpt_2, MAX)$ *if and only if* $lp1 \leq_{left} lpt2$ *holds.*

Please consult [Komuro] for the proofs of this and similar propositions concerning left and right endpoints.

## Sequencing

... now you have obtained a design of *Query Estimation*. You need to implement it efficiently.

**How can you efficiently and correctly implement the design?**

In order to get the real program we have to refine the previous definition considering the requirements of the system especially the efficiency issues. We should be careful in this refinement procedure to preserve the correctness we have proved.

- Consider the sequencing.

  Since *Initial Model* is written in terms of sets as is usual for the specification, it does not tell you anything about the order of the operations to implement *Query Estimation*. You should consider sequencing of the operations when implementing and optimizing the result of *Reification*.

**Therefore:**

Implement and optimize the design obtained by *Reification*. Make sure that the relation and constraints given in the model hold still true.

## Example of Sequencing

In the case of the database wrapper we have efficiency concern about the implementation of union and intersection. In fact the domain $D$ consists of a collection of intervals and naive implementation could compute over every possible combination of intervals. We solved this problem by transforming logical expression into disjunctive normal form and also sorting the intervals in Element with respect to the order of left endpoint. This may sound very natural. But there is one pitfall. When sorting the left endpoint, we have to use the ordering $\leq_{left}$ on left endpoints that is compatible with the ordering on intervals as described in Proposition 2. This ordering is different from the ordinary numerical ordering. If we use the numerical ordering, we may lose the correctness. In fact we can prove that we preserve the correctness if we use the ordering $\leq_{left}$.

Please consult [Komuro] for more detail.

## Concluding Remarks

### Formal Method

*Approximate Query Evaluation* is an example of development using formal method, where abstract interpretation [Hughes] is employed as the mathematical model. In the development using formal method we generally follow the same procedure as in the *Query Estimation*. That is, first construct *Initial Model*, then perform *Reification* and finally optimize the result by *Sequencing*.

### Query Subsumption

In the study of database there is a concept called query subsumption which means a weaker query than a given query. Query subsumption is used to generate a query for heterogeneous databases [Chang]. Although it is not explicitly stated, *Approximate Query Evaluation* is used in the procedure to get a query subsumption.

## References


[GOF] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.

[Hughes] J. Hughes. *Analysing strictness by abstract interpretation of continuations*, In Abstract Interpretation of Declarative Languages, pp.63-102 Ellis Horwood, 1987.

[Komuro] M. Komuro. *Application of Abstract Interpretation to Optimization of Database Access*, Proceedings of SEA Software Symposium 2001, pp.35-42 (in Japanese).

[Chang] K.C-C.Chang, H. Garcia-Molina, and A. Paepcke. *Boolean Query Mapping Across Heterogeneous Information Sources*, IEEE Transaction on Knowledge and Data Engineering, vol.8, No.4, 1996.